SproutCore Modular Loading Specification

Version 1.0
January 24, 2011

Authors:

Tom Dale tdale@apple.com

Majd Taby taby@apple.com

Requirements

Features

Asynchronous Code Loading

Developers can load JavaScript and CSS asynchronously at any point during the lifecycle of their application. This allows discrete units of functionality to be broken up and only loaded when needed. Dependencies are analyzed and loaded automatically.

Prefetching of Code

Code that is frequently used but not essential to application bootstrapping can be scheduled to be loaded in the background as soon as the user is idle. For example, a preference pane and its supporting code are not often used within the first few seconds of the application loading.

String Loading

The JavaScript code for a module can be loaded as a string that is held in memory until it is needed. That way, the initialization and memory cost is not paid until the code is actually used.

Specification

Overview

Developers can define modules of code that are loaded asynchronously after the initial payload of JavaScript, CSS and HTML. Each application must list the modules it needs in its Buildfile. Except for inlined modules, your application must explicitly load a module in order to make its code available.

Terminology

1. A ***module*** is a discrete collection of code and resources used within an application. It encapsulates a specific area of functionality.

2. A ***deferred module*** is a module that is loaded after the application has initialized. It must be explicitly requested by the application.

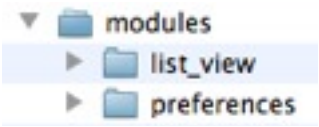
3. A ***prefetched module*** is a module that will be fetched automatically once the initial application payload has finished loading and executing.

4. An ***inlined module*** is a module that is included in the initial payload.

5. A ***framework*** serves as a structure upon which you build your application. Control flow is usually dictated by the framework. A framework may itself be composed of multiple modules.

Defining Modules

Modules are composed of JavaScript, CSS and image resources. Just like frameworks are contained in a `frameworks` directory, modules are contained in a `modules` directory. If you have two modules, `preferences` and `list_view`, the directory structure might look like this:



Apps and frameworks can contain modules, and modules can contain other modules.

Inside each module, the directory structure is the same as apps or frameworks. Controllers go in the `controllers` directory and CSS goes in the `resources` directory, for example.

Loading a module does not load its submodules automatically. If your module has dependencies on other modules, you can list them in the Buildfile and they will be loaded as soon as the primary module is loaded. Here's what it might look like in your Buildfile:

```
config 'modules/list_view',
  :required => ['sproutcore/desktop/collection']
```

Configuring Apps to Use Modules

Each application must list the modules it needs. You can do that using the `:deferred_modules`, `:prefetched_modules`, and `:inline_modules` directives. An to-dos application with several different modules might have a Buildfile like this:

```
config :todos,
  :deferred_modules   => ['todos/preferences', 'todos/group_view'],
  :prefetched_modules => ['todos/toolbar'],
  :inlined_modules    => ['todos/ui']
```

Note that, if you specify no deferred modules, prefetched modules, or inlined modules, SproutCore will assume your app is not "module aware" and will inline all modules, meaning that all code and resources will be sent in the initial download.

Loading Modules

Inside your application, you can tell a module to load by calling `SC.Module.loadModule(moduleName, target, action)`.

For example, if I wanted to load the 'todos/preferences' module and notify a controller:

```
SC.Module.loadModule('todos/preferences', ¬
'Todos.preferencesController', 'moduleDidLoad'))
```

This would start fetching the todos/preferences module, then call the `moduleDidLoad` method of the `Todos.preferencesController` once complete. Note that you can pass the target and action as either references of strings that are evaluated once the module has loaded. This makes it easy to handle the callbacks within the module itself.

Note: There is no need to manually load inlined modules. Their code and resources will have been loaded and evaluated before your application is loaded.